Sistemas Embarcados - Projeto Final (TP2)

Professor: Sérgio Johann Filho

Aluno: Bruno Bavaresco Zaffari

OBJETIVO DESTE TRABALHO

O objetivo deste trabalho foi implementar e validar um driver de dispositivo para criptografia baseada no algoritmo Triple DES (3DES) no sistema operacional UCX/OS, executando sobre o processador HF-RISC-E. Para isso, utilizou-se um núcleo criptográfico 3DES descrito em hardware (VHDL) e mapeado por MMIO no barramento do SoC.

A aplicação se comunica com o driver por meio das chamadas padrão write() (para criptografar ou descriptografar dados) e read() (para obter o resultado). O driver implementa controle de tarefas, verificação de permissões de acesso por Task ID, e possui suporte ao modo ECB (Electronic Code Book). Foram aplicados testes com mensagens reais e com gerenciamento de padding conforme o padrão PKCS#7.

Além disso, foram realizadas otimizações na comunicação com o IP criptográfico, incluindo verificação do sinal READY antes de ler a saída, evitando condições de corrida. A estrutura do driver permite modularidade para modos futuros (CBC, CTR) e garante integridade no processo de entrada e saída de dados criptografados. A aplicação principal foi usada para testar o pipeline completo de criptografia e descriptografia, validando a correção do algoritmo implementado.

AMARRAÇÃO DOS REGISTRADORES:

```
    process (clock_in, reset)
    begin
        if reset = '1' then
            key1_in_des <= (others => '0');
            key2_in_des <= (others => '0');
            key3_in_des <= (others => '0');
            data_in_des <= (others => '0');
            function_select_des <= '0';
            reset_des <= '0';
            ldkey_des <= '0';
            lddata_des <= '0';
        elsif clock_in'event and clock_in = '1' then
            if (ext_periph = '1' and data_we /= "0000") then
                case address(7 downto 4) is
                    when "0000" =>
                        function_select_des <= data_write_periph(4);
                        reset_des <= data_write_periph(3);
                        ldkey_des <= data_write_periph(2);
```

```
                    lddata_des <= data_write_periph(1);

                when "0001" =>      --  0xe7000010
                    key1_in_des(32 to 63) <= data_write_periph;
                when "0010" =>      --  0xe7000020
                    key1_in_des(0 to 31) <= data_write_periph;

                when "0011" =>      --  0xe7000030
                    key2_in_des(32 to 63) <= data_write_periph;
                when "0100" =>      -- 0xe7000040
                    key2_in_des(0 to 31) <= data_write_periph;

                when "0101" =>      -- 0xe7000050
                    key3_in_des(32 to 63) <= data_write_periph;
                when "0110" =>      -- 0xe7000060
                    key3_in_des(0 to 31) <= data_write_periph;

                when "0111" =>      -- 0xe7000070
                    data_in_des(32 to 63) <= data_write_periph;
                when "1000" =>      -- 0xe7000080
                    data_in_des(0 to 31) <= data_write_periph;
                when others =>
                    NULL;
            end case;
        end if;
    end if;
end process;


crypto_core: entity work.tdes_top
port map(
    key1_in => key1_in_des,
    key2_in => key2_in_des,
    key3_in => key3_in_des,
    function_select => function_select_des,
    data_in  => data_in_des,
    data_out => data_out_des,
    lddata => lddata_des,
    ldkey => ldkey_des,
    out_ready => out_ready_des,
    reset => reset_des,
    clock => clock_in
);
```

ESTRUTURAS DE DADOS CRIADAS:

```c
/* user data */
typedef enum { // SELECIONANDO O TIPO
    DECRYPT = 27,
    ENCRYPT = 54
}e_TYPE;

typedef enum { // SELECIONANDO O MODO
    ECB = 20,
    CBC = 40,
    CTR = 60
}e_MODE;

typedef struct {
    uint32_t KEY[6];
    e_TYPE  type;
    e_MODE  mode;
    uint32_t iv[2];
} s_CONFIG_tDES;

typedef struct {
    uint8_t *BUFFER; // INPUT no driver TDES
    size_t size;
} s_DATA_tDES;
```

```c
extern struct device_api_s dev_tdes;

void hex_dump_uint8(const uint8_t *data, size_t size);
```

Resumo do Trabalho com Trechos de Implementação:

O trabalho consistiu na implementação de um driver de dispositivo para um módulo de criptografia Triple DES (3DES) no sistema UCX/OS, rodando sobre o processador HF-RISC-E. O objetivo principal foi permitir que tarefas do sistema realizem operações de criptografia e descriptografia utilizando instruções simples de alto nível como write() e read().

Estrutura Geral e Inicialização:

Foi definido um conjunto de registradores para acesso às chaves, entrada e saída do bloco de criptografia, como por exemplo:

```
//============================================================================
// Definições e Funções de Baixo Nivel para Acesso ao Hardware
//============================================================================
#define DES_BASE            0xe7000000
#define DES_CONTROL         (*(volatile uint32_t *)(DES_BASE + 0x000))
#define DES_KEY1_1          (*(volatile uint32_t *)(DES_BASE + 0x010))
#define DES_KEY1_2          (*(volatile uint32_t *)(DES_BASE + 0x020))
#define DES_KEY2_1          (*(volatile uint32_t *)(DES_BASE + 0x030))
#define DES_KEY2_2          (*(volatile uint32_t *)(DES_BASE + 0x040))
#define DES_KEY3_1          (*(volatile uint32_t *)(DES_BASE + 0x050))
#define DES_KEY3_2          (*(volatile uint32_t *)(DES_BASE + 0x060))
#define DES_IN0             (*(volatile uint32_t *)(DES_BASE + 0x070))
#define DES_IN1             (*(volatile uint32_t *)(DES_BASE + 0x080))
#define DES_OUT0            (*(volatile uint32_t *)(DES_BASE + 0x090))
#define DES_OUT1            (*(volatile uint32_t *)(DES_BASE + 0x0A0))

#define DES_ENCRYPT         (1 << 4)
#define DES_DECRYPT         (0 << 4)
#define DES_RESET           (1 << 3)
#define DES_LDKEY           (1 << 2)
#define DES_LDDATA          (1 << 1)
#define DES_READY           (1 << 0)
#define BLOCKLEN_BYTES          8
#define BLOCKLEN_BITS           64
```

Para inicializar o driver e configurar chaves, foram utilizadas funções como:

Controle de Acesso e Estado

Foi implementado controle de uso exclusivo do driver por tarefa, usando o ucx_task_id() e um estado global:

```
#define NO_TASK 0xFFFF
static uint16_t tarefa_que_chamou = NO_TASK;

e_STATE state = OFF;
```

Esse controle impede que mais de uma tarefa use o driver simultaneamente e garante que apenas a tarefa que abriu o driver possa usá-lo.

Manipulação de Dados e Padding

A função tdes_write() recebe os dados da tarefa e realiza o pré-processamento, incluindo cálculo de padding PKCS#7 quando o tamanho do texto não é múltiplo de 8:

resto = count % BLOCKLEN_BYTES;

count_padding = (resto > 0) ? BLOCKLEN_BYTES - resto : 0;

Modos de Operação

A criptografia foi implementada para os modos ECB, CBC e CTR. O modo ECB simplesmente processa blocos isolados com dec/enc_block():

```
/// Tipo de operacao_____________________________________________
if (pconfig->mode == CTR)          ///CTR usa SEMPRE encrypt
    enc_str(pconfig);
else if (pconfig->type == ENCRYPT)
    enc_str(pconfig);
else
    dec_str(pconfig);
//_____________________________________________________________
```

ECB:

```
// Logida da escolha do modulo
for (size_t j = 0; j < QUANTITY_of_BLOCKS; ++j) {
    size_t base = j * BLOCKLEN_BYTES;
    memset(block, 0, sizeof(block));

    memcpy(&block[0], in_buf + base, 4);
    memcpy(&block[1], in_buf + base + 4, 4);

    if (pconfig->mode == ECB) {
        if(pconfig->type == ENCRYPT)
            enc_block(block);
        else
            dec_block(block);
    }
```

O modo CBC utiliza IV e XOR entre blocos:

```
else if (pconfig->mode == CBC) {
    if (pconfig->type == ENCRYPT) { // ENCRYPT

        block[0] ^= pconfig->iv[0];
        block[1] ^= pconfig->iv[1];
        enc_block(block);
        pconfig->iv[0] = block[0];
        pconfig->iv[1] = block[1];
    } else { // DECRYPT
        uint32_t next_iv[2] = {block[0], block[1]};
        dec_block(block);
        block[0] ^= pconfig->iv[0];
        block[1] ^= pconfig->iv[1];
        pconfig->iv[0] = next_iv[0];
        pconfig->iv[1] = next_iv[1];
    }
}
```

O modo CTR gera um keystream com incremento de IV:

```
}
else if (pconfig->mode == CTR) {
    keystream[0] = pconfig->iv[0];
    keystream[1] = pconfig->iv[1];
    /// Gera keystream cifrando o IV
    enc_block(keystream);
    block[0] ^= keystream[0];/// XOR do bloco de entrada com o keystream
    block[1] ^= keystream[1];/// XOR do bloco de entrada com o keystream
    /// Incrementa o contador
    pconfig->iv[1]++;
    if (pconfig->iv[1] == 0) pconfig->iv[0]++;
}

memcpy(out_buf + base, block, BLOCKLEN_BYTES);
```

Alocação Dinâmica e Buffer de Saída

A alocação de buffers foi feita dinamicamente com malloc e liberada ao final do processo:

```
        memcpy(out_buf + base, block, BLOCKLEN_BYTES);
    }

    pconfig->iv[0] = temp_IV[0];
    pconfig->iv[1] = temp_IV[1];
    //hex_dump_uint8(out_buf, count_total);// DEGUG
    memcpy(pdata->BUFFER, out_buf, count_total);
    free(in_buf);
    free(out_buf);

    NOSCHED_LEAVE();
    return count_total;
}
```

A saída é copiada para o buffer do dispositivo para ser recuperada via read() posteriormente:

Validação Final:

```
    printf("------------------------------- FIM -------------------------------\n");

    if (memcmp(plain_text, texto_retorno_leg, strlen(plain_text)) == 0) {
        printf("[VERIFICACAO]: PASSOU Texto decifrado BATE com o original.\n");
    } else {
        printf("[VERIFICACAO]: Texto decifrado nao confere com o original.\n");
    }

    free(texto_cifrado);
    free(texto_retorno_leg);
    _delay_us(3);
    // FECHANDO -------------------------------------
    dev_close(&dev);
    // FECHANDO -------------------------------------
}
```

O driver foi testado com a frase "**the quick brown fox jumps over the lazy dog**" e validou corretamente os blocos e o padding aplicado. Todo o processo de criptografia e descriptografia se mostrou funcional.

Observações e Ajustes

Durante o desenvolvimento, foi tentada a implementação de uma função personalizada de Hexdump, mas a tentativa causou comportamento indefinido que bloqueava o funcionamento correto do sistema. Posteriormente percebeu-se que o problema não estava na função original do professor, mas em interações com a seleção da libc, que coincidiram com os testes do Hexdump. Por isso, decidiu-se utilizar a versão original fornecida como referência. Apesar de atrasar um pouco o progresso, esse episódio ajudou a isolar problemas e entender melhor a interação entre bibliotecas e funções auxiliares no UCX/OS.

APP:

```
#include <ucx.h>
#include <device.h>
#include "tdes_driver.h"
//As memorias foram postas com tamanho fixo mas pode ser mudadas
static size_t count=0;
char plain_text[50] = {0};
char texto_cifrado[50]= {0};
char texto_retorno_leg[50]= {0};
char print[50]= {0};

//-----------------------------------------------------------------
static s_CONFIG_tDES config = {
    .KEY  = {0x88880001, 0x88880002, 0x88880003, 0x88885000, 0x88882000,
0x88883000},
    .type = ENCRYPT,
    .mode = ECB,
    .iv   = {0x88880001, 0x88880000}
```

```
};

static s_DATA_tDES dados_enc = {
    .BUFFER= NULL,    // INPUT no driver TDES
    .size= 0
};

static s_DATA_tDES dados_dec = {
    .BUFFER= NULL,    // INPUT no driver TDES
    .size= 0
};

static struct device_s dev = {
    .name = "tdes_ECB",
    .config = &config,
    .data = &dados_enc,
    .api = &dev_tdes
};
//-----------------------------------------------------

//Funcoes auxiliares ---------------------------------
size_t calculo(char *mensagem, size_t modo) {
    // modo == 0 é Debug
    if (!mensagem) {
        printf("[APP_MAIN] Mensagem nula\n");
        return 0;
    }
    size_t BLOCKLEN_BYTES = 8;

    size_t tamanho = strlen((const char *)mensagem);

    size_t blocos_cheios     = tamanho / BLOCKLEN_BYTES;
    size_t resto             = tamanho % BLOCKLEN_BYTES;
    size_t padding_necessario = (resto > 0) ? (BLOCKLEN_BYTES - resto) : 0;
    size_t blocos_totais     = blocos_cheios + (resto > 0 ? 1 : 0);
    size_t tamanho_total     = blocos_totais * BLOCKLEN_BYTES;

    if (modo == 0){
        printf(" [APP_MAIN] Pre-calculo:\n");
        printf("- Tamanho original:  %u bytes\n", tamanho);
        printf("- Blocos completos:        %u\n", blocos_cheios);
        printf("- Bytes de padding:        %u\n", padding_necessario);
        printf("- Total final (criptografado): %u bytes (%u blocos)\n",
tamanho_total, blocos_totais);
    }
    return tamanho_total;
}
```

```
//-------------------------------------------------------------------------
------------


void func_encode_decode(void){
    strcpy(plain_text, "the quick brown fox jumps over the lazy dog");

    count = calculo(plain_text, 0);

    printf("------------------------------ INICIO -------------------------
--------\n");
    printf("\n\n-----------------------------------------------------------
-----\n \
    MODO DE OPERACAO: <<<<<<<<<<<EBC>>>>>>>>>>>>\n \
    ------------------------------------------------------------------\n\n");
    printf("[APP_MAIN] TEXTO ORIGINAL: \n%s\n", plain_text);
    hexdump(plain_text, count);
    printf("\n");

    // ABRINDO ----------------------------------------------------------
--------
    if (dev_open(&dev, 0)) {
        printf("task %d: driver in use.\n", ucx_task_id());
        return;
    }
    // ABRINDO ----------------------------------------------------------
--------
    printf("\n");

    // CIFRANDO ---------------------------------------------------------
--------
    dev_write(&dev, plain_text, count);
    dev_read(&dev, texto_cifrado, count);
    printf("[APP_MAIN] TEXTO CIFRADO: \n");
    memset(print, 0, sizeof(char)*50);
    memcpy(print, texto_cifrado, count);
    hexdump(print, count);

    // DECIFRANDO -------------------------------------------------------
----------
    config.type = DECRYPT;
    dev.data = &dados_dec;
    _delay_us(3);
    dev_write(&dev, texto_cifrado, count);
    dev_read(&dev, texto_retorno_leg, count);
    printf("[APP_MAIN] TEXTO RECEBIDO DECIFRADO: \n");
    memset(print, 0, sizeof(char)*50);
    memcpy(print, texto_retorno_leg, count);
    hexdump(print, count);
```

```
    printf("-------------------------------- FIM ---------------------------
-----\n");

    if (memcmp(plain_text, texto_retorno_leg, strlen(plain_text)) == 0) {
        printf("[VERIFICACAO]: PASSOU Texto decifrado BATE com o
original.\n");
    } else {
        printf("[VERIFICACAO]: Texto decifrado nao confere com o
original.\n");
    }

    free(texto_cifrado);
    free(texto_retorno_leg);
    _delay_us(3);
    // FECHANDO --------------------------------------
    dev_close(&dev);
    // FECHANDO --------------------------------------
}

void task_encode_decode(void){
    func_encode_decode();
    krnl_panic(0);
    for(;;);
}
int32_t app_main(void){
    ucx_task_spawn(task_encode_decode, DEFAULT_STACK_SIZE);
    dev_init(&dev);
    return 1;
}
```

VALIDACÕES:

```
13
14  ------------------------------------------------------------------
15  |  |  |  |  |  |  MODO DE OPERACAO: <<<<<<<<<<<CBC>>>>>>>>>>>>
16  |  |  |  |  ------------------------------------------------------------
17  --
18
19  [APP_MAIN] TEXTO ORIGINAL:
20  the quick brown fox jumps over the lazy dog
21
22  40003580                         74 68 65 20 71 75 69 63  |........the
23  quic|
24  40003590 6B 20 62 72 6F 77 6E 20  66 6F 78 20 6A 75 6D 70  |k brown fox
25  jump|
26  400035A0 73 20 6F 76 65 72 20 74  68 65 20 6C 61 7A 79 20  |s over the l
27  azy |
28  [TDES_DRIVER]: Usuario atual: 65535
29  [TDES_DRIVER]: Usuario que chamou: 0
30
31  [APP_MAIN] TEXTO CIFRADO:
32
33  400034E0                                     26 E5 91 17  |.....5.@.5.@
34  &...|
35  400034F0 42 35 AD 89 3D 39 46 DD  99 5F 06 89 FA 6C F9 AA  |B5..=9F.._...
36  .l..|
37  40003500 7B 15 FC E1 C7 8A 71 93  51 8A D0 00 8B B5 8E 27  |{.....q.Q...
38  ...'|[APP_MAIN] TEXTO RECEBIDO DECIFRADO:
39
40  400034E0                                     74 68 65 20  |.0...5.@.5.@
41  the |
42  400034F0 71 75 69 63 6B 20 62 72  6F 77 6E 20 66 6F 78 20  |quick brown
43  fox |
44  40003500 6A 75 6D 70 73 20 6F 76  65 72 20 74 68 65 20 6C  |jumps over t
45  he l|-------------------------------- FIM ----------------------------
46  ----
47  [VERIFICACAO]: PASSOU Texto decifrado BATE com o original.
48  [TDES_DRIVER]: Closing driver
```

```
13
14  ----------------------------------------------------------------
15                      MODO DE OPERACAO: <<<<<<<<<<<<CTR>>>>>>>>>>>>>
16                  ----------------------------------------------------------------
17  -
18
19  [APP_MAIN] TEXTO ORIGINAL:
20  the quick brown fox jumps over the lazy dog
21
22  40003580                          74 68 65 20 71 75 69 63  |........the
23  quic|
24  40003590 6B 20 62 72 6F 77 6E 20  66 6F 78 20 6A 75 6D 70  |k brown fox
25  jump|
26  400035A0 73 20 6F 76 65 72 20 74  68 65 20 6C 61 7A 79 20  |s over the l
27  azy |
28  [TDES_DRIVER]: Usuario atual: 65535
29  [TDES_DRIVER]: Usuario que chamou: 0
30
31  [APP_MAIN] TEXTO CIFRADO:
32
33  400034E0                                      09 1C DC 84  |.....5.@.5.@
34  ....|
35  400034F0 1F 88 E3 41 B4 C3 32 6C  D5 B8 AA 76 3F 5A 74 F8  |...A..2l...v
36  ?Zt.|
37  40003500 EB F6 E6 BD 6B C2 BC 11  FF B7 E7 F6 EB 03 3B FD  |....k.......
38  ..;.|[APP_MAIN] TEXTO RECEBIDO DECIFRADO:
39
40  400034E0                                      74 68 65 20  |./...5.@.5.@
41  the |
42  400034F0 71 75 69 63 6B 20 62 72  6F 77 6E 20 66 6F 78 20  |quick brown
43  fox |
44  40003500 6A 75 6D 70 73 20 6F 76  65 72 20 74 68 65 20 6C  |jumps over t
45  he l|-------------------------------- FIM ----------------------------
46  ----
47  [VERIFICACAO]: PASSOU Texto decifrado BATE com o original.
48  [TDES_DRIVER]: Closing driver
```

```
                                                                  
------------------------------------------------------------------
                    MODO DE OPERACAO: <<<<<<<<<<<<EBC>>>>>>>>>>>>
            ------------------------------------------------------------------
-

[APP_MAIN] TEXTO ORIGINAL:
the quick brown fox jumps over the lazy dog

40003510             74 68 65 20  71 75 69 63 6B 20 62 72  |....the quic
k br|
40003520 6F 77 6E 20 66 6F 78 20  6A 75 6D 70 73 20 6F 76  |own fox jump
s ov|
40003530 65 72 20 74 68 65 20 6C  61 7A 79 20 64 6F 67 00  |er the lazy
dog.|
[TDES_DRIVER]: Usuario atual: 65535
[TDES_DRIVER]: Usuario que chamou: 0

[APP_MAIN] TEXTO CIFRADO:

40003470                         8D 26 F6 0B 82 C1 A9 AA  |h5.@h5.@.&..
....|
40003480 6D E0 11 15 2B 1D EF 99  54 1D 8E 06 78 AC D9 25  |m...+...T...
x..%|
40003490 D7 32 22 C5 85 96 20 7A  1B 23 78 6B B7 F1 6B 91  |.2"... z.#xk
..k.|[APP_MAIN] TEXTO RECEBIDO DECIFRADO:

40003470                         74 68 65 20 71 75 69 63  |h5.@h5.@the
quic|
40003480 6B 20 62 72 6F 77 6E 20  66 6F 78 20 6A 75 6D 70  |k brown fox
jump|
40003490 73 20 6F 76 65 72 20 74  68 65 20 6C 61 7A 79 20  |s over the l
azy |--------------------------------- FIM ----------------------------
----
[VERIFICACAO]: PASSOU Texto decifrado BATE com o original.
[TDES_DRIVER]: Closing driver
```